Compare dynamic programming and standard recursion by filling out the table below.

Algorithm	*Top-down or bottom-up?*	*Solve the same subproblem once?*	*Always solve all sub-problems*
Dynamic Programming	bottom up	yes	yes
Standard Recursion	bottom up	yes	yes

Problem 2 (6 points) 5/6

Let $A_1, \ldots, A_4$ be matrices with dimensions 5×2, 2×4, 4×5, 5×10, respectively. In finding an optimal parenthesization of the matrix chain product $A_1{*}A_2{*}A_3{*}A_4$, we use two tables **m[,]** and **s[,]** below. Here **m[i,j]** stores the optimal cost of computing subchain $A_i..A_j$ and **s[i,j]** records the index k where the optimal parenthesization splits $A_i..A_j$ between A_k and A_{k+1} for some k with $i \le k \le j-1$.

a- What is the recursive equation of **m[i,j]**?

$$M[i,j] = \min_{i \le k < j} \left\{ M[i,k] + M[k+1,j] + P_{i-1} \times P_k \times P_j \right.$$

b- Fill the empty entries in the two tables. Show your work in each case.

m

j \ i	1	2	3	4
4	~~240~~ 244	141	200	0
3	90	40	0	
2	40	0		
1	0			

s

j \ i	1	2	3
4	1	3	3
3	1	2	0
2	1	0	

الحل خلف الورقة

c- Now, give the optimal parenthesization of the matrix chain product $A_1{*}A_2{*}A_3{*}A_4$. Show how you came up with the solution using the tables above.

(A1)((A2 A3)(A4))

explain:- using the table(s) above selected the place of the parentasis for example by using s[1,6] i managed to know that i have to put the parantheses after A1 isolating it from the rest.

Problem 3 (6 points) 6/6

Solve the following instance of the Knapsack Problem using Dynamic programming paradigm. The maximum allowed weight is Wmax = 10.

i	1	2	3	4
V_i	20	10	15	31
W_i	6	1	2	5

a- Give the recursive equation you used to define the data structure needed for your dynamic programming solution. Then, fill the proposed data structure.

	i \ w	0	1	2	3	4	5	6	7	8	9	10
	0	0	0	0	0	0	0	0	0	0	0	0
$v_i=20$, $w_i=6$	1	0	0	0	0	0	0	20	20	20	20	20
$v_i=10$, $w_i=1$	2	0	10	10	10	10	10	20	30	30	30	30
$v_i=15$, $w_i=2$	3	0	10	15	25	25	25	25	30	35	45	45
$v_i=31$, $w_i=5$	4	0	10	15	25	25	31	41	46	56	56	56

$$m[i,w] = \begin{cases} m[i-1,w] & w_i > w \\ & \text{otherwise} \\ \max(m[i-1,w-w_i]+v_i,\ m[i-1,w]) & \end{cases}$$

```
Knapsak{
~~while{~~
let R = n , let solution[1,w]
let D = W
while(R ≤ i){

if (m([R,W] == m[R-1,W])
  R = R-1; }

else {

solution[1]= n[R,W]

m[R,D] = m[R-1, D-W_i]  }
```

b- What is your solution to this instance of the Knapsack problem?

(4,3,2)

$w_4 + w_3 + w_2 = 5 + 2 + 1 = 8$

$v_4 + v_3 + v_2 = 31 + 15 + 10 = 56$

c- Write a pseudo-code to find the items of the optimal solution.

```
Knapsak{
  for i = 0 -> n{
    M[i,0] = 0 }
  for w = 0 -> n{
    M[0,w] = 0 }
  for i = 1 -> n{
  for w = 1 -> n{
  if w < M[i,w].wi {
    M[i,w] = M[i-1,w] }
  else
  M[i,w] = max[m[i-1,w-wi]+vi, m[i-1,w]]
```

```
Knapsak{
for ...{
if(M[i,w] == M[i-1,w]){
```

~~خلف الورقة~~
خلف ورقة السابقة

-1	2	-19	-5	4	-7	-51	-2	-22	-13	-15	36

Problem 4 (5 points) 3/5

The Longest Decreasing Subsequence problem is defined as follows: Given a sequence of *n* real numbers A[1]... A[n], determine a subsequence (not necessarily contiguous) of maximum length in which the values in the subsequence form a strictly decreasing sequence.

Example:

The length of the longest decreasing Subsequence in [-1, 2, -19, -5, 4, -7, -51, -2, -22, -13, -15, 36] is 5.

a- Give the pseudo-code of a Dynamic programming algorithm that solves the Longest Decreasing Subsequence problem.

```
LDS{
while(i≤n){
    if A[i]≥A[i-1]{
    i=i+1 }
  else{
   count++;
   if(count≥S[i-1]){
   S[i]←count}
   else{
   S[i]=S[i-1]} }
          }
return S[]; }
```

– – 2

b- What is the time complexity of your algorithm? Prove it!

$O(n)$

$T(n) = 6n + 1$

$6n + 1 \leq 7n$

$c = 7$

$n_0 = 1$